2025-01-13 03:11:21

configs

```
{'_end': '2025-01-10:02',
 '_start': '2025-01-10:01',
 'exc': 'bybit',
 'params': {'latencies': {'ack_private': 0,
                          'ack_public': 0,
                          'feed_private': 0,
                          'feed_public': 0,
                          'req_private': 0,
                          'req_public': 0},
            'maker_fees': 0.0,
            'taker_fees': 0.0},
 'tickers': ['SOLUSDT']}
```

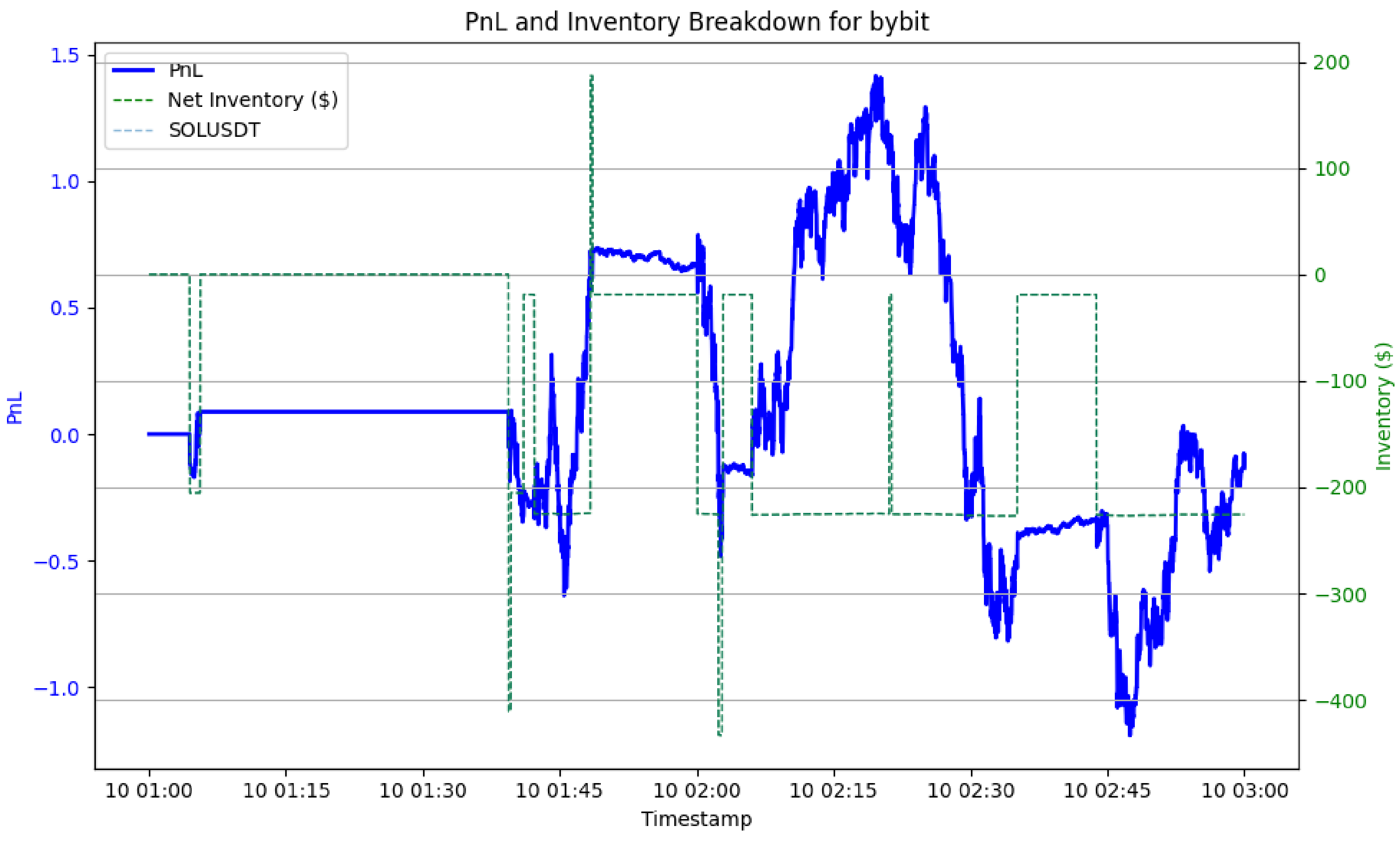
stats

```
{'binance': None,
 'bybit': {'#trades': 59,
           '0s Markouts': 0s   -0.000015
0s   -0.000015
dtype: float64,
           '11s Markouts': -7.368662532822541e-06,
           'Tick Markouts': -1.540771245685202e-05,
           'flip_win_ratio': 0.6,
           'interval_quartiles': ['0s', '0s', '11s'],
           'inventory_flips': 5,
           'max_leverage': 0.04,
           'max_norm_beta': 0.04,
           'maxdd(%)': 0.03,
           'pnl': -0.14,
           'roi': -0.0,
           'sharpe': -7.76,
           'terminal': 9999.86,
           'turnover': 0.39,
           'volume': Decimal('3899.41'),
           'volume/s($)': Decimal('0.65')}}
```

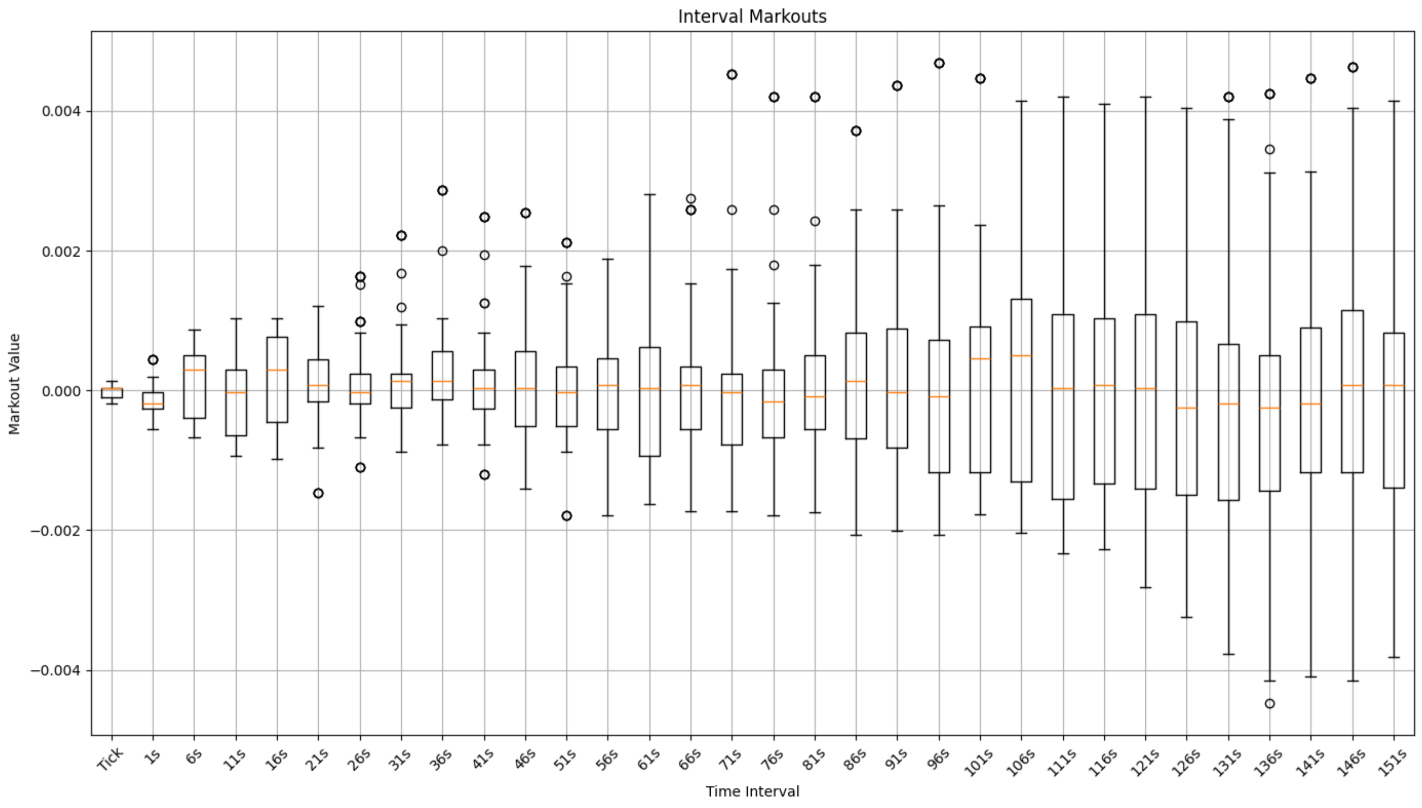
perf[inventory]

```
                            equity  inventory    pnl  SOLUSDT
ts
2025-01-10 01:00:00.877  10000.000      0.000  0.000    0.000
2025-01-10 01:00:01.877  10000.000      0.000  0.000    0.000
2025-01-10 01:00:02.877  10000.000      0.000  0.000    0.000
2025-01-10 01:00:03.877  10000.000      0.000  0.000    0.000
```

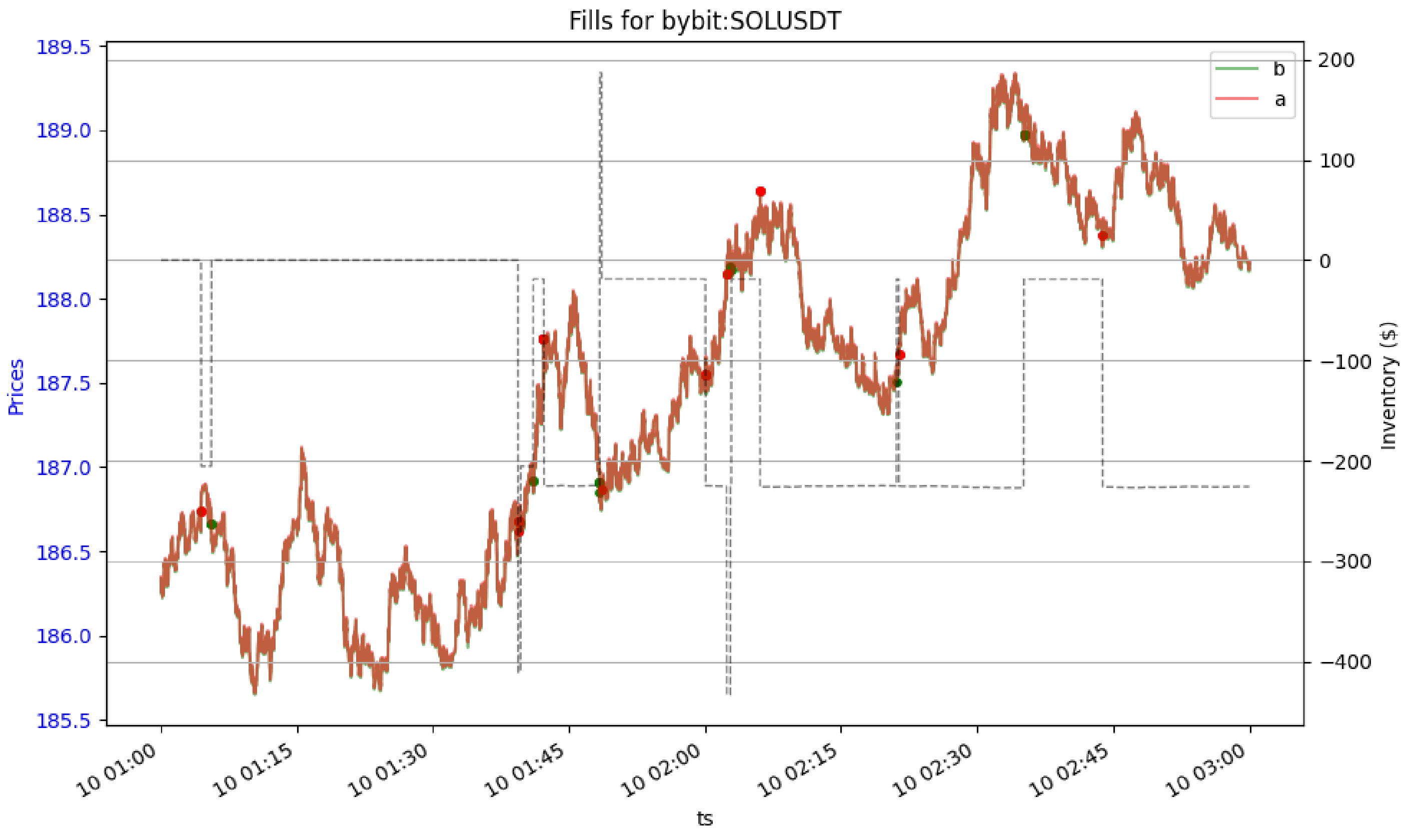
```
2025-01-10 01:00:04.877  10000.000      0.000  0.000    0.000
...                            ...        ...    ...      ...
2025-01-10 02:59:55.877   9999.877   -225.846 -0.123 -225.846
2025-01-10 02:59:56.877   9999.925   -225.798 -0.075 -225.798
2025-01-10 02:59:57.877   9999.925   -225.798 -0.075 -225.798
2025-01-10 02:59:58.877   9999.889   -225.834 -0.111 -225.834
2025-01-10 02:59:59.877   9999.865   -225.858 -0.135 -225.858

[7200 rows x 4 columns]
```



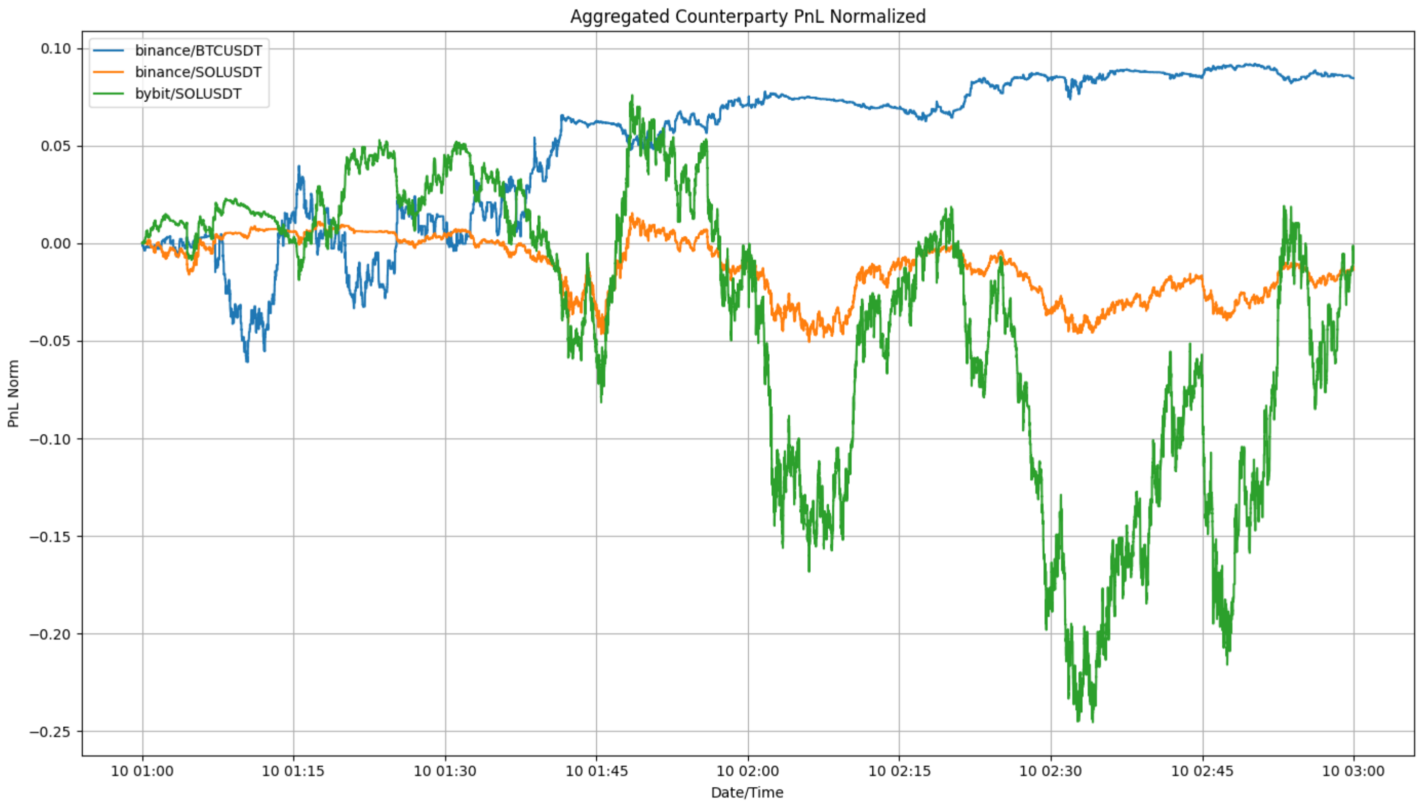
markouts



fills



counterparty_pnl


Aggregated Counterparty PnL Normalized
binance/BTCUSDT
binance/SOLUSDT
bybit/SOLUSDT
PnL Norm
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.25
10 01:00
10 01:15
10 01:30
10 01:45
10 02:00
10 02:15
10 02:30
10 02:45
10 03:00
Date/Time

Script

```
import os
import pytz
import asyncio
import logging
import argparse
import numpy as np
from pprint import pprint
from decimal import Decimal
from datetime import datetime
from dotenv import load_dotenv
load_dotenv()

import quantpylib.utilities.math as math
import quantpylib.standards.markets as markets

from quantpylib.hft.oms import OMS
from quantpylib.hft.feed import Feed
from quantpylib.gateway.master import Gateway
from quantpylib.utilities.general import _time,pdf_snapshot

from utils import get_key,restore_archives

show = True
parser = argparse.ArgumentParser(description="Parse script mode.")
parser.add_argument('mode', type=lambda arg:int(arg), help="Set script mode.")
args = parser.parse_args()
simulated = False if args.mode == 1 else True

config_keys = {
    "binance":{},
    "bybit":{
        'key':'BYBIT_DEMO_KEY',
        'secret':'BYBIT_DEMO_SECRET',
        'env':'BYBIT_DEMO_ENV'
    },
}

exc = 'bybit'
tickers = ["SOLUSDT"]
stream_data = {
    "binance":["BTCUSDT","SOLUSDT"],
    "bybit":["SOLUSDT"],
}
reference = {
    'SOLUSDT':('binance','SOLUSDT'),
}

run = None
time = None
replayer = None
print_interval = 120 * 1000
```

```python
if simulated:
    from quantpylib.hft.mocks import Replayer,Latencies
    start='2025-01-10:01'
    end='2025-01-10:02'
    latencies={
        Latencies.REQ_PUBLIC:0,
        Latencies.REQ_PRIVATE:0,
        Latencies.ACK_PUBLIC:0,
        Latencies.ACK_PRIVATE:0,
        Latencies.FEED_PUBLIC:0,
        Latencies.FEED_PRIVATE:0,
    }
    replayer_configs = {
        "maker_fees":0.0,
        "taker_fees":0.000,
        "latencies":latencies,
    }

else:
    run = lambda : asyncio.sleep(1e9)
    time = lambda : _time()

gateway = Gateway(config_keys=get_key(config_keys))

async def quote_markets(replayer,oms,feed,ticker):
    live_orders = oms.orders_peek(exc=exc)
    live_positions = oms.positions_peek(exc=exc)

    order_value = 200

    ref_exc,ref_ticker = reference[ticker]
    reference_trade_id = await feed.add_trades_feed(
        exc=ref_exc,
        ticker=ref_ticker,
        buffer=500,
    )
    reference_trades = feed.get_feed(reference_trade_id)
    reference_l2_id = await feed.add_l2_book_feed(
        exc=ref_exc,
        ticker=ref_ticker,
        depth=20,
        buffer=100,
    )
    reference_lob = feed.get_feed(reference_l2_id)

    ticker_trade_id = await feed.add_trades_feed(
        exc=exc,
        ticker=ticker,
        buffer=500,
    )
    ticker_trades = feed.get_feed(ticker_trade_id)

    last_print = time()
```

```python
async def l2_handler(lob):
    nonlocal last_print
    stamp = time()
    stamp = stamp - (stamp % print_interval)
    if stamp != last_print:
        print(datetime.fromtimestamp(time() / 1000,tz=pytz.utc))
        last_print = stamp

    pos_amount = live_positions.get_ticker_amount(ticker=ticker)
    inventory_notional = float(pos_amount) * lob.get_mid()
    q = inventory_notional / order_value

    order_bids = live_orders.get_bid_orders(ticker=ticker)
    order_asks = live_orders.get_ask_orders(ticker=ticker)

    tds = reference_trades.get_sample(n=15)
    notional = 20000 if len(tds) == 0 else np.sum(tds[:,1] * tds[:,2])
    ref_vamp = reference_lob.get_vamp(notional)
    if np.isnan(ref_vamp):
        return

    bid = min(
        ref_vamp - ((6 + np.tanh(q) * 4) * 1e-4) * ref_vamp,
        lob.get_bids()[1,0]
    )
    ask = max(
        ref_vamp + ((6 + np.tanh(-q) * 4) * 1e-4) * ref_vamp,
        lob.get_asks()[1,0]
    )

    bid_price = Decimal(str(oms.rounded_price(exc=exc,ticker=ticker,price=bid)))
    ask_price = Decimal(str(oms.rounded_price(exc=exc,ticker=ticker,price=ask)))

    orders = []
    if not any(bid.price == bid_price for bid in order_bids):
        orders.append({
            "exc":exc,
            "ticker":ticker,
            "amount":order_value/lob.get_mid(),
            "price":bid_price,
            "round_to_specs":True,
        })
    if not any(ask.price == ask_price for ask in order_asks):
        orders.append({
            "exc":exc,
            "ticker":ticker,
            "amount":order_value/lob.get_mid() * -1,
            "price":ask_price,
            "round_to_specs":True,
        })

    cancels = []
    for order in order_bids:
        if order.price != bid_price:
                cancels.append({
                    "exc":order.exc,
                    "ticker":order.ticker,
                    "cloid":order.cloid
                })
        for order in order_asks:
            if order.price != ask_price:
                cancels.append({
                    "exc":order.exc,
                    "ticker":order.ticker,
                    "cloid":order.cloid
                })
        if orders:
            try:
                await asyncio.gather(*[
                    oms.limit_order(**order) for order in orders
                ])
            except e:
                logging.exception(e)
        if cancels:
            try:
                await asyncio.gather(*[
                    oms.cancel_order(**cancel) for cancel in cancels
                ])
            except:
                logging.exception(e)

    l2_feed = await feed.add_l2_book_feed(
        exc=exc,
        ticker=ticker,
        depth=20,
        buffer=100,
        handler=l2_handler
    )

async def hft(replayer,oms,feed):
    btc_feed = await feed.add_trades_feed(
        exc='binance',
        ticker='BTCUSDT',
        buffer=500,
    )

    await asyncio.gather(*[
        quote_markets(replayer=replayer,oms=oms,feed=feed,ticker=ticker)
        for ticker in tickers
    ])
    await run()

    if simulated:
        await sim_report(replayer)

async def sim_report(replayer):
    folder = './logs/reports'
    exchange_plot = f'{folder}/exchange.png'
    markouts_plot = f'{folder}/markouts.png'
    prices_plot = '{folder}/prices_{ticker}.png'
    counterparty_plot = f'{folder}/counterparty_pnl.png'
    fairprices_plot = f'{folder}/fairprices.png'

    report_data = []
    report_data.append((
        'configs',
        {
            'exc':exc,
            'tickers':tickers,
            '_start':start,
            '_end':end,
            'params':replayer_configs,
        },
        'dict'
    ))

    statistics = replayer.statistics()
    pprint(statistics)
    report_data.append(('stats',statistics,'dict'))

    df_exchange = replayer.df_exchange(exc=exc,save=exchange_plot,show=show)
    report_data.append(('perf[inventory]',df_exchange,'df'))
    report_data.append(('perf[inventory]',exchange_plot,'img'))

    df_markouts = replayer.df_markouts(save=markouts_plot,show=show)
    report_data.append(('markouts',markouts_plot,'img'))

    for ticker in tickers:
        save = prices_plot.format(folder=folder,ticker=ticker)
        df_prices = replayer.df_prices(
            ticker=ticker,exc=exc,show=show,save=save
        )
        report_data.append(('fills',save,'img'))

    norm_pnls = replayer.df_counterparty_pnl(exc_tickers=stream_data,save=counterparty_plot,show=show)
    report_data.append(('counterparty_pnl',counterparty_plot,'img'))



    pdf_snapshot(
        save=f'{folder}/{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}.pdf',
        code_file=__file__,
        report_data=report_data,
        include_comments=False
    )

async def sim_prepare():
    l2_data = {exchange:{} for exchange in stream_data}
    trade_data = {exchange:{} for exchange in stream_data}

    for exchange,tickers in stream_data.items():
        await asyncio.gather(*[
            restore_archives(
                exc=exchange,
                ticker=ticker,
                depth=20,
                start=start,
                end=end,
            ) for ticker in tickers
        ])

        lob_archives = [
            Feed.load_lob_archives(
                exc=exchange,
                ticker=ticker,
                depth=20,
                start=start,
                end=end
            ) for ticker in tickers
        ]

        trade_archives = [
            Feed.load_trade_archives(
                exc=exchange,
                ticker=ticker,
                start=start,
                end=end
            ) for ticker in tickers
        ]

        l2_data[exchange] = {ticker:lob_archive for ticker,lob_archive in zip(tickers,lob_archives)}
        trade_data[exchange] = {ticker:trade_archive for ticker,trade_archive in zip(tickers,trade_archives)}

    global replayer, run, time
    replayer = Replayer(
        l2_data=l2_data,
        trade_data=trade_data,
        gateway=gateway,
        **replayer_configs
    )
    oms = replayer.get_oms()
    feed = replayer.get_feed()
    run = lambda : replayer.play()
    time = lambda : replayer.time()
    return oms, feed

async def main():
    await gateway.init_clients()
    if simulated:
        oms,feed = await sim_prepare()
    else:
        oms = OMS(gateway=gateway,exchanges=[exc],refresh_orders_snapshot=10,refresh_positions_snapshot=10)
        feed = Feed(gateway=gateway)

    await oms.init()
    await hft(replayer,oms,feed)
    await gateway.cleanup_clients()
```

```
if __name__ == '__main__':
    asyncio.run(main())
```